LA-UR-25-23479

Approved for public release; distribution is unlimited.

Title:	Driving Continuous Integration and Developer Workflows with Spack
Author(s):	Berger, Richard Felix
Intended for:	Spack User Meeting, 2025-05-07/2025-05-08 (Chicago, Illinois, UNITED STATES)
Issued:	2025-04-10

Los Alamos
NATIONAL LABORATORY

Driving Continuous Integration and Developer Workflows with Spack

Richard Berger

May 8, 2025

Outline

Motivation

Driving CMake with Spack

Spack Deployments

Outline

Motivation

Driving CMake with Spack

Spack Deployments

What are we building and testing?

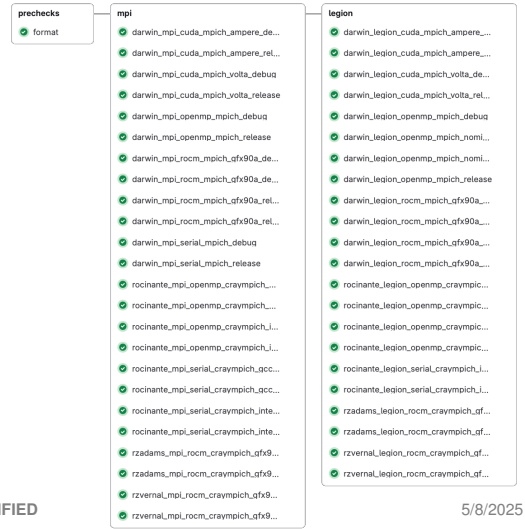
- C/C++/Fortran HPC codes
- Research and Production codes
- Run on Tri-Lab (LANL, LLNL, Sandia) systems
  - LANL: Crossroads / Rocinante / Chicoma / Venado
  - LLNL: El Capitan / Tuolumne / Sierra
- CPU-only and GPU-accelerated systems

What are we building and testing?

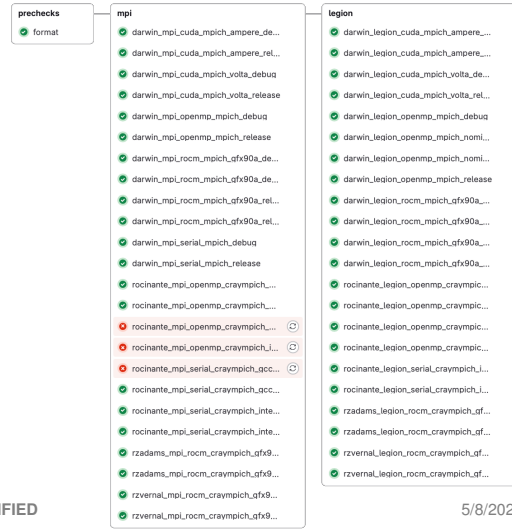
- Builds: Release, Debug
- Code Variants / Library Configurations
- Compilers: GCC, Clang, Intel, NVIDIA, AMD
- Accelerations: OpenMP, CUDA, ROCm
- Communication:
  - MPI (MPICH, OpenMPI, CrayMPICH)
  - GASNET
- Hardware
  - GPUs: NVIDIA Volta/Ampere, AMD MI250/MI300A
  - Interconnects: HPE/Cray Slingshot 11, Infiniband

Continuous Integration (CI) Pipelines

- GitLab for Source Code Management
- Pipelines for CI
  - main branch
  - scheduled (nightly, weekly)
  - merge requests
- Jacamar CI Runners for TriLab systems
  - jobs can run via batch system



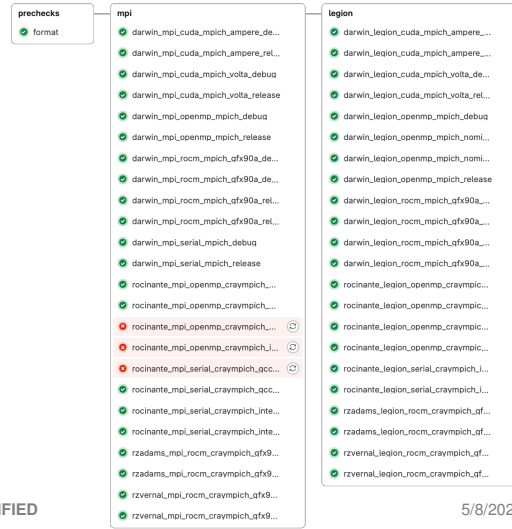
CI Failures: What now?



CI Failures: What now?

Developers

“How do I recreate that CI job?”



Our answer

Every GitLab job, on **all clusters**, prints a message like this at the beginning:

```
################################################################################
To recreate this CI run, follow these steps:

ssh rzadams
cd /your/code/checkout
flux alloc -N=1 --time-limit=2h
source .gitlab/build_and_test.sh rzadams craympich-rocm-gfx942-gcc

See 'source .gitlab/build_and_test.sh -h' for more options.
################################################################################
```

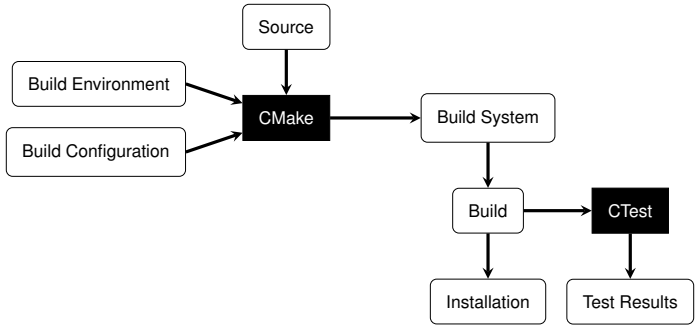
Outline

Motivation

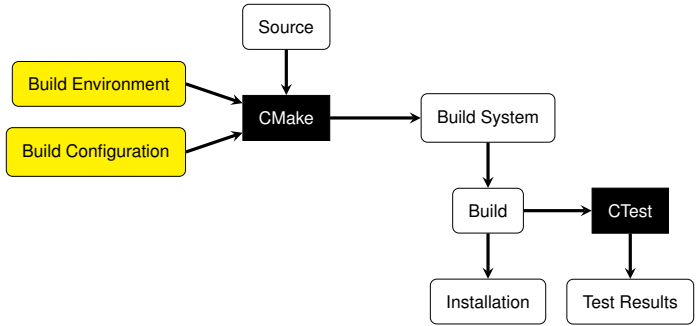
Driving CMake with Spack

Spack Deployments

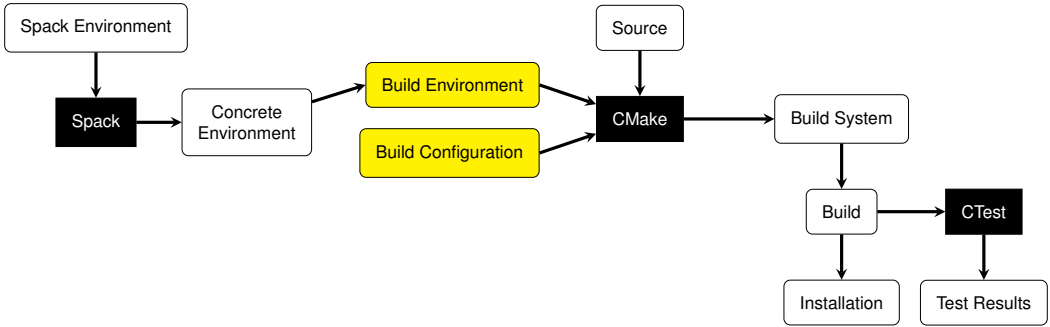
CMake Workflow



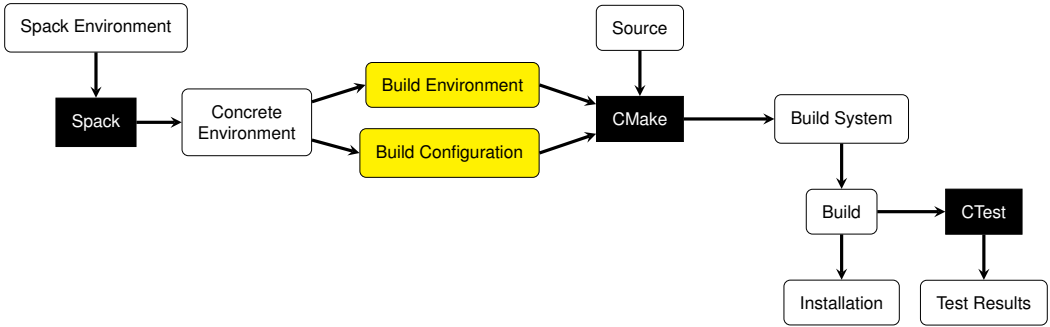
CMake Workflow



CMake Workflow



CMake Workflow



Requirement: Spack package for our code

```
class MyPackage(CMakePackage):
  git = "https://github.com/namespace/myproject.git"

  version("main", branch="main")

  variant("boost", default=False)
  variant("kokkos", default=False)

  depends_on("cmake@3.20:", type="build")
  depends_on("boost@1.80:", when="+boost")
  depends_on("kokkos@4:", when="+kokkos")
  depends_on("googletest@1.12:", type="test")

  ...
```

- How to get the source code
- Which versions exist
- Which configuration options exist
- What are dependencies, given these configuration options and other factors

Requirement: Spack package for our code

```
class MyPackage(CMakePackage):
  ...

  def cmake_args(self):
    args = [
      self.define_from_variant("ENABLE_BOOST", "boost"),
      self.define_from_variant("ENABLE_KOKKOS", "kokkos"),
      self.define("ENABLE_CUDA", self.spec.satisfies("^kokkos+cuda")),
      self.define("BUILD_TESTING", self.run_tests),
    ]
    return args
```

Given a **build configuration** and an **environment**, what are the options for our build system?

Spack Environments = Environment + Build Configuration

```yaml
# minimal/spack.yaml
spack:
  spec:
  - mypackage
  packages:
    mypackage:
      require:
       - "+boost"
```

```yaml
# ampere/spack.yaml
spack:
  spec:
  - mypackage
  packages:
    mypackage:
      require:
       - "+boost +kokkos"
    kokkos:
      require:
       - "+cuda cuda_arch=80"
```

Installing our code with Spack

```
# activate environment defined in myenv/spack.yaml
spack env activate -d myenv

# install our software
spack install
```

⚠ Note: This is **not** what we want for CI and Development!

Configuring your current project checkout with Spack

```
spack env activate -d myenv

spack develop -p ${SOURCE_DIR} -b ${BUILD_DIR} --no-clone myproject@main
spack concretize -f
spack install --include-build-deps --only dependencies

spack install --test root --until cmake myproject
```

Spack Environment

Configuring your current project checkout with Spack

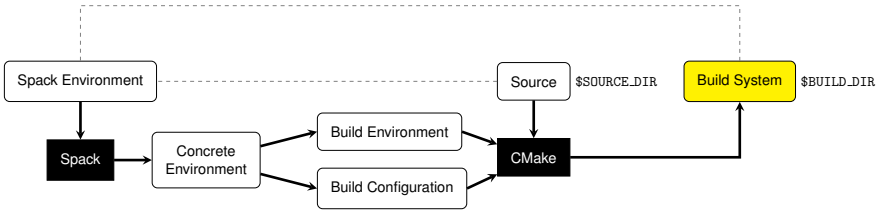
```
spack env activate -d myenv

spack develop -p ${SOURCE_DIR} -b ${BUILD_DIR} --no-clone myproject@main
spack concretize -f
spack install --include-build-deps --only dependencies

spack install --test root --until cmake myproject
```

Configuring your current project checkout with Spack

```
spack env activate -d myenv

spack develop -p ${SOURCE_DIR} -b ${BUILD_DIR} --no-clone myproject@main
spack concretize -f
spack install --include-build-deps --only dependencies

spack install --test root --until cmake myproject
```

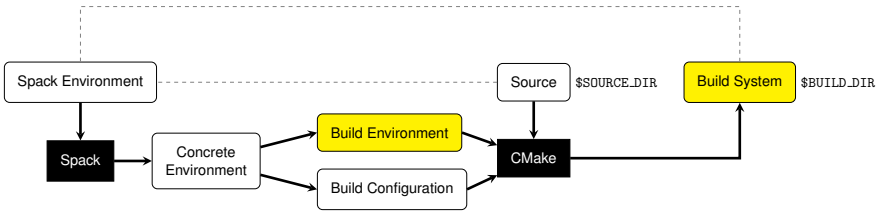
Configuring your current project checkout with Spack

```
spack env activate -d myenv

spack develop -p ${SOURCE_DIR} -b ${BUILD_DIR} --no-clone myproject@main
spack concretize -f
spack install --include-build-deps --only dependencies

spack install --test root --until cmake myproject
```



Getting the Spack Build Environment

```
# interactive sub-shell
spack build-env myproject bash

# sourceable shell script
spack build-env --dump ${BUILD_DIR}/build_env.sh myproject
```



Using the Spack Build Environment

```
# activate build environment
source ${BUILD_DIR}/build_env.sh

# undo Spack CMake default of verbose Makefiles
cmake -DCMAKE_VERBOSE_MAKEFILE=off ${BUILD_DIR}

# build code
cmake --build ${BUILD_DIR} --parallel
```

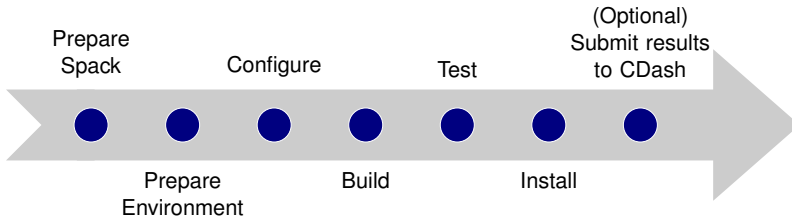
CI script helpers

```
# simplified version of helper in build_and_test.sh
cmake_build() {
  (
    # activate build environment
    source ${BUILD_DIR}/build_env.sh

    # undo Spack CMake default of verbose Makefiles
    cmake -DCMAKE_VERBOSE_MAKEFILE=off ${BUILD_DIR}

    # build code
    cmake --build ${BUILD_DIR} --parallel
  )
}
```

CI script helpers

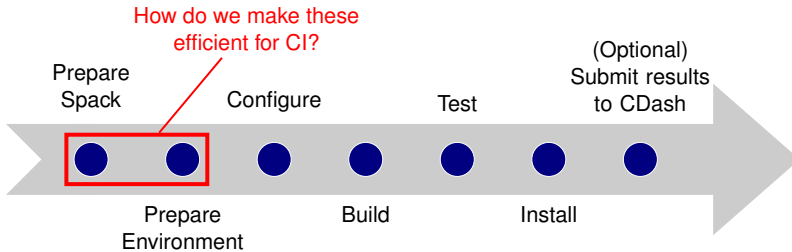
```
# simplified short versions of helpers in build_and_test.sh
prepare_env() {
  spack activate -d ${SPACK_ENV_NAME}
  spack develop -p $SOURCE_DIR -b ${BUILD_DIR} --no-clone myproject@main
  spack concretize -f
  spack install --include-build-deps --only dependencies -j $(nproc) myproject
}

spack_cmake_configure() {
  spack install --test root -u cmake myproject
  spack build-env --dump ${BUILD_DIR}/build_env.sh myproject
}
```

CI Job Steps



- each step is a bash shell function and can be invoked manually
- CI just goes through all of them in a fixed order
- developers's can stop at any point, make modifications, and continue via the shell functions.

CI Job Steps



- each step is a bash shell function and can be invoked manually
- CI just goes through all of them in a fixed order
- developers's can stop at any point, make modifications, and continue via the shell functions.

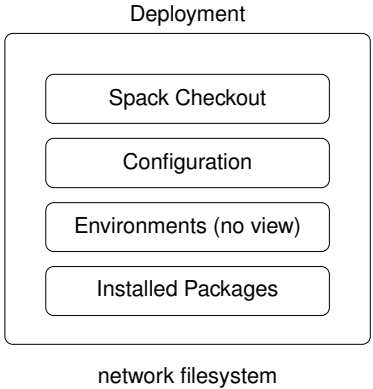
Outline

Motivation

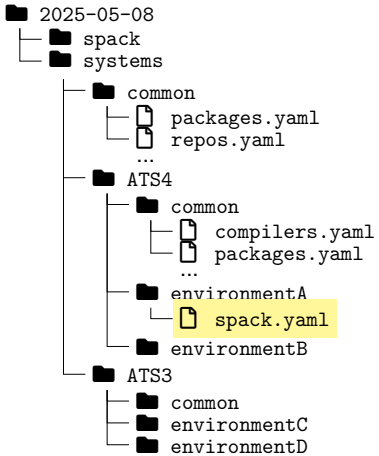
Driving CMake with Spack

Spack Deployments

Spack Deployments

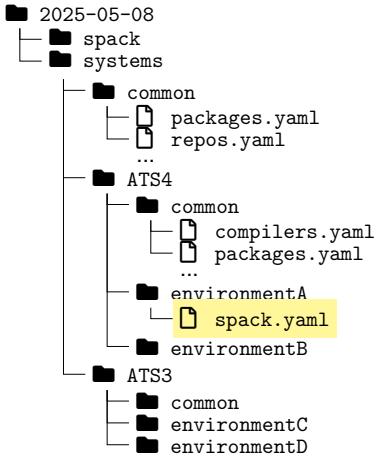


Spack Environments with shared configuration




```yaml
spack:
  include:
  - $spack/../systems/common/packages.yaml
  - $spack/../systems/common/repos.yaml
  - $spack/../systems/ATS4/common/compilers.yaml
  - $spack/../systems/ATS4/common/packages.yaml
  specs:
    - myproject
  view: true
  concretizer:
    unify: true
  packages:
    kokkos:
      require:
       - "+rocm amdgpu_target=gfx942"
```

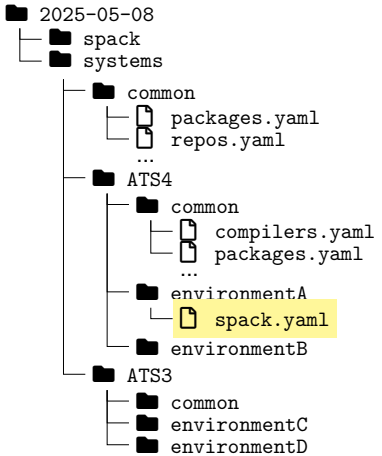
Spack Environments with shared configuration



project-specific configuration

```yaml
spack:
  include:
  - $spack/../systems/common/packages.yaml
  - $spack/../systems/common/repos.yaml
  - $spack/../systems/ATS4/common/compilers.yaml
  - $spack/../systems/ATS4/common/packages.yaml
  specs:
    - myproject
  view: true
  concretizer:
    unify: true
  packages:
    kokkos:
      require:
       - "+rocm amdgpu_target=gfx942"
```

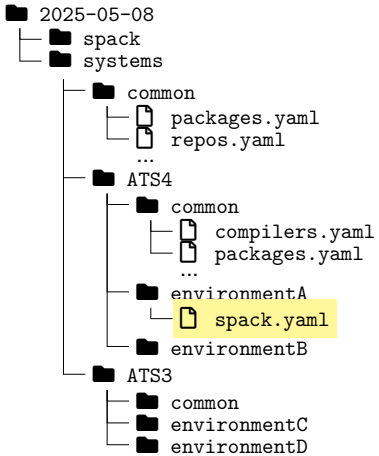
Spack Environments with shared configuration




```yaml
spack:
  include:
  - $spack/../systems/common/packages.yaml
  - $spack/../systems/common/repos.yaml
  - $spack/../systems/ATS4/common/compilers.yaml
  - $spack/../systems/ATS4/common/packages.yaml
  specs:
    - myproject
  view: true
  concretizer:
    unify: true
  packages:
    kokkos:
      require:
       - "+rocm amdgpu_target=gfx942"
```

system-specific configuration

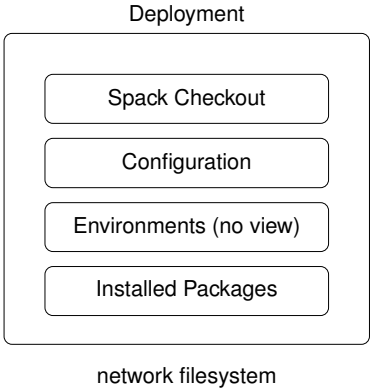
Spack Environments with shared configuration



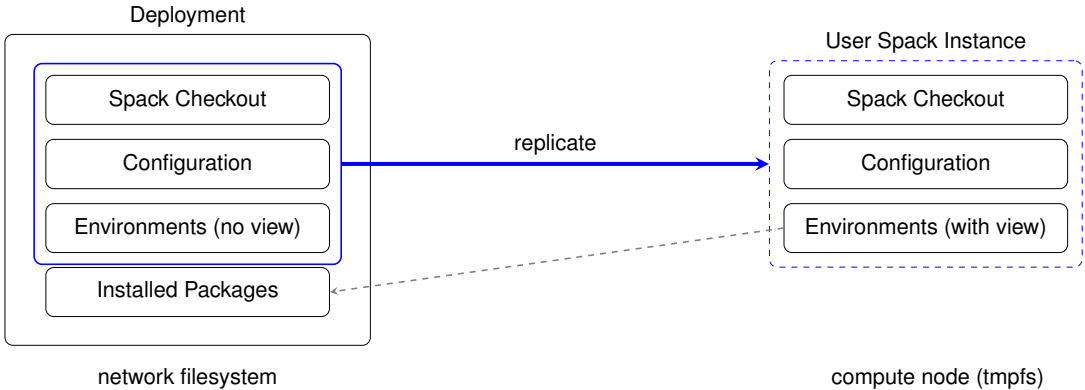

```yaml
spack:
  include:
  - $spack/../systems/common/packages.yaml
  - $spack/../systems/common/repos.yaml
  - $spack/../systems/ATS4/common/compilers.yaml
  - $spack/../systems/ATS4/common/packages.yaml
  specs:
    - myproject
  view: true
  concretizer:
    unify: true
  packages:
    kokkos:
      require:
       - "+rocm amdgpu_target=gfx942"
```

environment-specific configuration

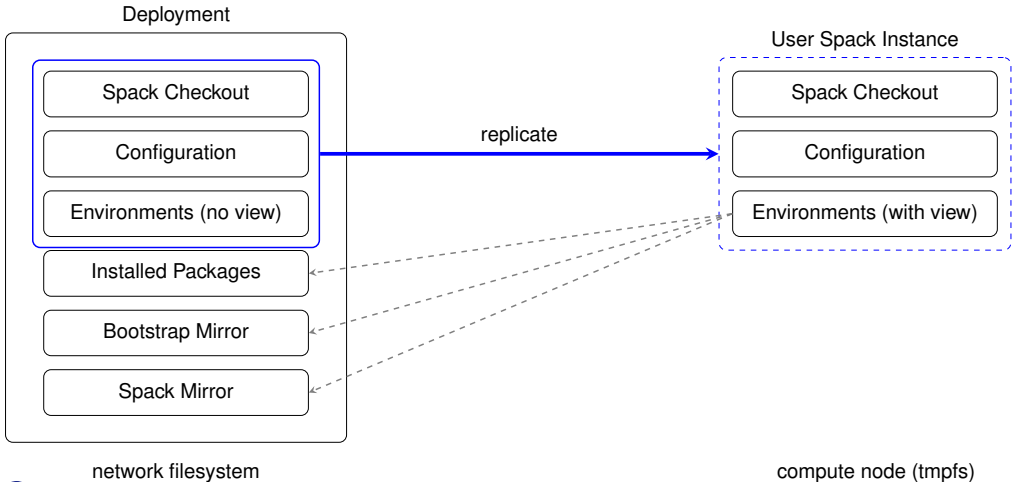
Spack Deployments



Spack Deployments



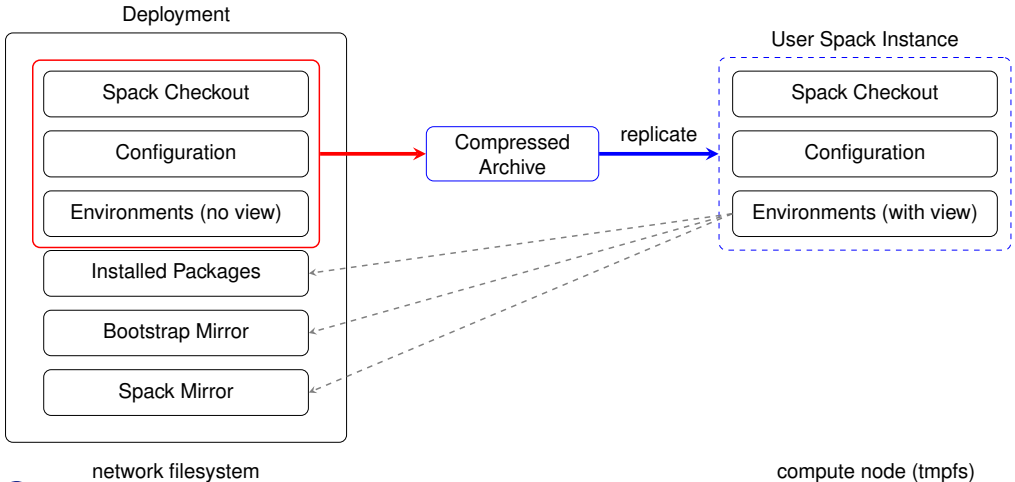
Bootstrap & Spack Mirrors



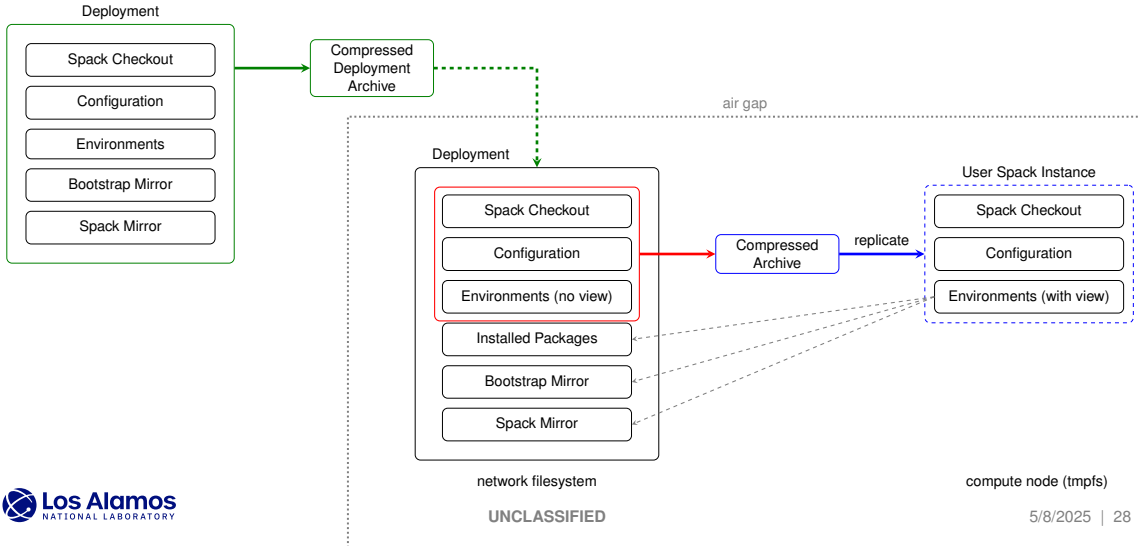
Spack Environments with upstream configurations

```
spack:
  include:
  - $spack/../systems/common/packages.yaml
  - $spack/../systems/common/repos.yaml
  - $spack/../systems/ATS4/common/bootstrap.yaml # generated, pointing to original
  - $spack/../systems/ATS4/common/compilers.yaml
  - $spack/../systems/ATS4/common/packages.yaml
  - $spack/../systems/ATS4/common/upstreams.yaml # generated, pointing to original
  - $spack/../systems/ATS4/common/mirrors.yaml   # generated, pointing to original
  specs:
    - myproject
  view: true
  concretizer:
    unify: true
  packages:
    ...
```

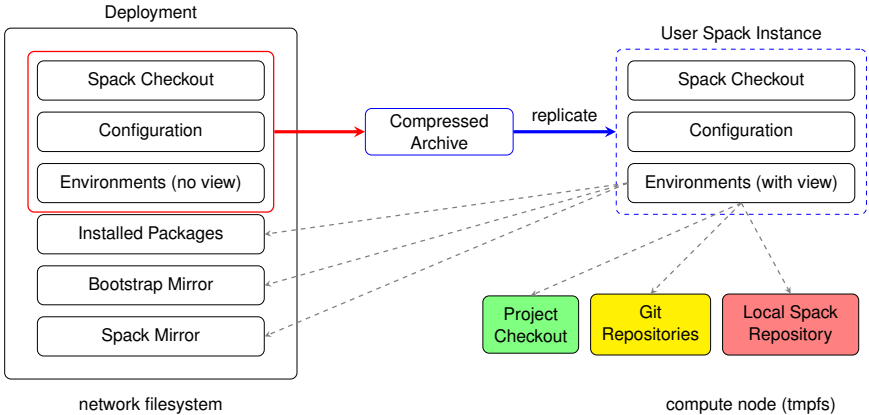
Optimizing Replication



Deploying on air-gapped systems



What Developers and the CI see



Summary

Deployment Pipeline

- places read-only shared Spack instance at known location at each target system
- create bootstrap and package source mirrors for all environments
- prebuilds software dependencies of all environments on all target systems

Continuous Integration Pipelines / Developer Workflows

- utility script drives entire build and test process for both CI and Development
- script creates a (temporary) Spack instance that uses a deployment as upstream
- Spack environments are used to encode and create the build environment and build configuration

Questions